Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing

Junwei Cao, Darren J. Kerbyson, and Graham R. Nudd
High Performance Systems Laboratory, Department of Computer Science, University of Warwick, Coventry, CV4 7AL, U. K.
{junwei, djke, grn}@dcs.warwick.ac.uk

Abstract

Resource management is an important infrastructure in the grid computing environment. Scalability and adaptability are two key challenges in the implementation of such complex software systems. In this work we introduce a new model for resource management in a metacomputing environment using a hierarchy of homogeneous agents that has the capability of service discovery. The performance of the agent system can be improved using different combinations of optimisation strategies. A modelling and simulation environment has been developed in this work that enables the performance of the system to be investigated. A simplified model of the resource management infrastructure is given as a case study and simulation results are included that show the impact of the choice of performance optimisation strategies on the overall system performance.

1. Introduction

A Computational Grid requires both hardware and software infrastructures to provide dependable, consistent, pervasive, and inexpensive access to high-end computational capability [11,13]. An ideal grid environment should provide access to the available resources in a seamless manner, which requires many design features. These include administrative hierarchy, communication services, information services, naming services, resource management and scheduling, security and authorization etc [2].

The overall aim of the resource management is to efficiently schedule applications that need to utilize the available resources in the metacomputing environment. In essence, it should turn a radically heterogeneous environment into a virtual homogeneous one, which introduces two key challenges:

Scalability. The grid may potentially encompass all high performance computing resources. A given component of the grid will have it's own functions, resources, and environment. These are not necessarily geared to work together in the overall grid. They may be physically located in different organizations and may not be aware of each other.

Adaptability. A grid is a dynamic environment where the location, type, and performance of the components are constantly changing. For example, a component resource may be added to, or removed from, the grid at any time. These resources may not be entirely dedicated to the grid, hence their computational capabilities will vary over time.

Resource management and scheduling infrastructures must be provided to address these challenges. Existing techniques cannot meet the requirements of both scalability and adaptability simultaneously. In this work, we present a new model for the resource management infrastructure using a hierarchy of homogeneous agents which has the capability of service discovery.

An agent-based hierarchical model is used to address the problem of the scalability, which is an extension of our previous work [6]. Software agents have been accepted to be a powerful high-level abstraction for the modelling of complex software systems [12].

In this work, an agent is considered to be both a service provider and a service requestor. Service is an important concept in many distributed computing and communication infrastructures (e.g. Jini [1], Bluetooth [3]). We use service here to describe the details of a resource within the grid. Resource management, scheduling, and allocation can be abstracted to the processes of service *advertisement* and service *discovery*.

Performance issues arise when service is advertised and discovered in a large-scale multi-agent system. Different optimisation strategies can be used to improve the system efficiency. A performance modelling and simulation environment has been developed that enables the performance of the system to be investigated. A case study, that models a simplified grid environment, is included. Simulation results show the impact of the choice of different performance optimisation strategies on the efficiency of the resource management infrastructure.

This work is based on existing resource management and prediction capabilities of the PACE toolset [4,5]. The motivation to develop a Performance Analysis and Characterization Environment (PACE) is to provide quantitative data concerning the performance of sophisticated applications running on local high performance resources [15]. It is envisaged that the agent infrastructure developed here will be amalgamated with the capabilities of PACE for grid environments.

Several solutions have been offered that address to some extent the issues of resource management and scheduling. Our work is different from these in a number of ways. Some of the principle existing work is described below.

- Condor [16] – uses the matchmaker/entity structure (which can be both provider and requestor), in which the matchmaker becomes the bottleneck of the system Achieving scalability is difficult, and focuses more on protocol issues than on performance issues.
- Globus [8] – uses a Metacomputing Directory Service (MDS) [10], which adopts the data representations and API defined by the LDAP service [18]. It is designed to meet requirements of both scalability and dynamic resources. Performance issues have not been a key consideration in its implementation.
- Hector [17] – uses a master/slave allocator to manage the dynamic resources, in which the issue of scalability is not addressed.
- Legion [9] – uses a Resource Management Infrastructure (RMI) in which the information collection is very similar to the information service in Globus. Objects are used as the main system abstraction throughout, whereas we use agents as a high level abstraction.
- NetSolve [7] – uses an agent as a database and a resource broker to implement resource management. The coordination of different agents are not considered in detail.
- Ninf [14] – the metaserver component monitors multiple computing servers on the network, and performs scheduling and load balancing of client requests (similar to the agents in the NetSovle).

The rest of the paper is organised as follows: In Section 2 the homogeneous agent hierarchy is presented. In Section 3, the metrics used for the performance evaluation of this hierarchy, and the different optimisation strategies available are described. In Section 4 the simulation environment is described and results from a case study concerning a small grid environment are presented.

2. Agent-Based Hierarchical Model

In this section a homogeneous agent-based hierarchical model is introduced as the basis of understanding the service advertisement and discovery mechanisms that can be used to implement the functions of resource management and scheduling.

The hierarchical model is illustrated in Figure 1. There is a single type of the component, the agent, which is used to compose the whole system. This is an abstraction of the computing resource in the software infrastructure of the resource management. Each agent has the same set of functions. They can send requests and provide services. Every agent can act as a router between a request and a service. In Figure 1 different terms are used to differentiate the level of the agent in the hierarchy. The broker is an agent that heads the whole hierarchy, maintaining all service information of the system. A coordinator is an agent that heads a sub-hierarchy. A leaf-node is actually termed an agent in this description.

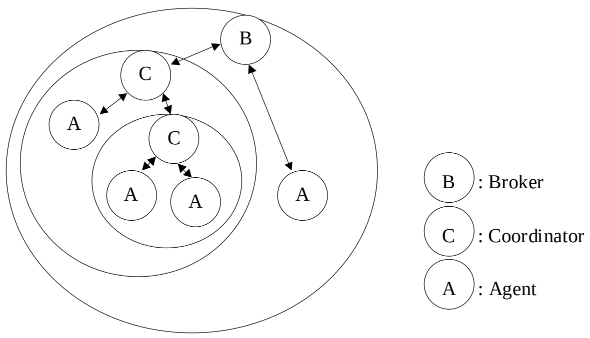


Figure 1. Agent hierarchy

When a new agent wants to join the system, in the hierarchical model, it will broadcast to find its nearest existing agent. An agent can only have one connection to an agent higher in the hierarchy to register with, but be registered with many lower level agents. All requests that enter a sub-hierarchy must arrive at the coordinator of the sub-hierarchy first and then dispatched to the lower agents. From the view of service providers, a sub-hierarchy can be regarded just as an agent.

If an agent has the required service information, it can contact the target agent directly. Otherwise, it must search its local agents, or ask its upper agent, for a service discovery (ie. to find an agent that can provide the requested service). The lower or upper agent can also ask other agents for assistance until the service information is found. The agent can then connect directly to the target and directly request the service. All connections between the agents are broken after use of the service is finished.

The services offered by an agent can change over time. When this occurs, the corresponding service information needs also to be updated. The dynamics of this system increases the difficulty of resource management and allocation. The essential issue is how an agent advertises its services and also coordinates with other agents to discover the required services in the most efficient way.

Service Advertisement. The service information of an agent can be advertised in the hierarchy (both up and down). Different strategies can be used to decide when, and how, to advertise a service but with different performances. This is discussed in greater detail in Section 3.

Service Discovery. When an agent requests a service, it will first check its own knowledge to see if it is already aware of the service. If it is, it will contact the target agent directly. Otherwise it may contact its upper agent until the available service is found.

The main data type for the advertisement and discovery is the Agent Capability Table (ACT). Different strategies can use different kinds of ACTs. The process of the service advertisement and discovery corresponds to the maintenance and lookup of the ACTs. The basic structure of an ACT item consists of two parts: agent identity and service information. The service information is system dependent. For example, simple service information can include: the service name, its performance, its lifetime, its scope, and so on. Complex service information can also include the service interface information. However, as mentioned above, this work focuses on performance issues.

Two extreme situations can be considered:

1. No service advertisement – results in complex service discovery. In this situation no ACTs are maintained in the agents. Each agent has no knowledge of the services offered by other agents. When a service is request, a service discovery process is required which may be complex and traverse a large number of agents in the system. The service information is pulled from the agents at discovery time, and so this is a pure *data-pull* model.
2. Full service advertisement – requires no service discovery. In this situation, each agent advertises to all the other agents. Hence each agent has complete knowledge on the available services in the system and no discovery process is required. When a request is made, the service is found in any agents ACT. The service information has been pushed to the other agents during the advertising processes, and so this is a pure *data-push* model.

Different systems can use different optimisation strategies to achieve high performance. For example in static systems, where the frequency of change in the service information is far less than the frequency of service request, the pure data-push model can be used to achieve high performance service discovery. In extremely dynamic systems, where the frequency change in the service information is far greater than the request frequency, the pure data-pull model can be used to achieve high performance. Most practical systems will have characteristics in-between these two extremes. The performance issues are discussed in greater detail in Section 3.

3. Service Discovery Performance

In this section we introduce the basic performance issues in the service discovery, since we view the resource management infrastructure as an agent-based hierarchical system and the function implementation of the scheduling and allocation as the process of the service advertisement and discovery. Some common performance metrics are given first and different optimisation strategies are discussed next.

3.1. Service Discovery Metrics

There are kinds of performance criteria that can be used to describe the service discovery performance part of the model. What is considered as high performance depends on the system requirements. However, there are some common characteristics of the system that are usually a concern of the system developer. These include:

Discovery Speed

Each request from an agent can pass one or more agents in order to find a target agent that can provide the required service. The performance of the discovery process is mainly based on the number of routing connections, since the size of data communication is small. Fewer connections has a quick discovery process, and the higher system performance. In the whole system, there may be simultaneous service requests. The average service discovery speed, v is defined as:

$$v = \frac{r}{d} \tag{1}$$

where r is the total number of requests during a certain period, and d is the total number of connections made for the discovery.

System Efficiency

The cost for the service discovery also includes connections made for service advertisement and data maintenance. Service advertisement may add additional workload to the system. For each request to find a corresponding service, the total number of connections, c, between agents includes those for the discovery processes, d, and also those for the advertising processes, a.

$$c = d + a \tag{2}$$

The efficiency of the system can be considered as the ratio of the total number of requests, r, during a certain period, to the total number of connections c.

$$e = \frac{r}{c} \tag{3}$$

Load Balancing

In some of the systems when the system resources are critical, load-balancing may be an important issue. In this system, no agents are used only for service discovery. There is no reason to have any agent with a higher discovery workload than any other. For a system with n agents, the workload, w_k, of each can be described as

$$w_k = o_k + i_k \qquad (k = 1......n) \tag{4}$$

where o_k and i_k are the outgoing and incoming connection times. We can use the mean square deviation of the w_k to describe the load balancing level of the system, b:

$$b = \sqrt{\frac{\Sigma_k \left(w_k - \overline{w}\right)^2}{n}} \qquad \text{where} \qquad \overline{w} = \frac{\Sigma_k w_k}{n} \tag{5}$$

Success Rate

In some of the performance optimisation strategies the discovery model cannot guarantee to find the target service (that may actually exist in the system). However, in a general system a reasonable service discovery success rate should always be achieved. The success rate, f, describes successful service discovery:

$$f = \frac{r_f}{r} \times 100\% \tag{6}$$

Most of the time, these service discovery metrics are conflictive, that is not metrics can be high at the same time. For example, a quick discovery speed does not mean high efficiency, as sometimes quick discovery may be achieved through the high workload encountered in service advertisement and data maintenance, leading to low system efficiency. It is necessary to the critical factors of the practical system, and then to use the different performance optimisation strategies to reach high performance.

3.2. Performance Optimisation Strategies

There are several kinds of performance optimisation strategies that can be considered. Most have been used in current practical systems with an assumption that the performance can be optimised. The effects of these strategies are not discussed in detail especially when the dynamics of the system increases. The combination of these strategies may also lead to a different performance.

Use of cache

Caching previous service discovery results is a good strategy for performance optimisation that assumes a request may be required more than once. Cached service information is expressed as C_ACT. When an agent sends a request for service discovery, the result can be stored in C_ACT, and hence looked up when next requested. If however the service has changed and is not available any more, the agent may update the C_ACT and perform another service discovery.

Many current network applications use caches to optimise performance. Using cached service information may result in direct service discovery in one step. Another advantage of using cache is that it adds no additional data maintenance workload. However, if the service information changes frequently compared to the request frequency, using cache may decrease the service discovery speed. So the efficiency of using cache depends on the characteristics of the actual system.

Using local and global knowledge

Adding some local or global knowledge to an agent is also a performance optimisation that assumes that services are often required by local agents. A request may need less connections to find the local service as the higher-level agents need not take part in the discovery process. The system load can also be reduced.

In order to coordinate the agents to find the services, two kinds of ACTs can be used in each agent to record the service details and information, which are local (L_ACT) and global (G_ACT). Each agent has one L_ACT to record the service information about the agents registered with it. If a request is within the capabilities of the local agents, the agent may directly dispatch the request to the target agent. The G_ACT in an agent is actually a copy of its upper agent's L_ACT. Thus an agent can have the information of more services and be able to contact them directly without submitting the request to the upper agent.

Unlike the C_ACT, additional data maintenance workload is needed for the L_ACT and G_ACT. There are basically three ways to maintain their contents. Firstly, the agent itself can go to pull the corresponding data directly. For L_ACT, the agent can ask its lower agents for their L_ACT, and for G_ACT, the agent can ask its upper agent for its L_ACT. Secondly, the maintenance of the L_ACT and G_ACT can also be driven by changes in service. If contents in one L_ACT of an agent are changed, it may report this to the L_ACT of its upper agent, and may also inform the change to the G_ACTs of its lower agents. Thirdly, the L_ACT and G_ACT can also be updated in the same way as the C_ACT.

The process for the service discovery using the L_ACT and G_ACT is also different from that using the C_ACT. When an agent receives a request, it will look up its L_ACT. If the agent finds that one of its lower agents can provide the service, it will dispatch the request directly to the corresponding agent. Otherwise, it will look up its G_ACT. If G_ACT shows that another agent can provide the service, it will dispatch the request to that agent. If the service is not found in either, the agent will ask its upper agent for further service information. After the upper agent returns the result, it can update its own G_ACT and return the result to the agent who originated the request.

Limit service lifetime

Another performance optimisation strategy is adding a service lifetime limitation to the attributes of the service information. This lifetime should be pre-estimated before the service is advertised. The agent can check the ACTs frequently and delete out-of-date service information. This can avoid unnecessary routing processes and increase the speed of service discovery. There is also no additional data maintenance workload. However, the lifetime of some services in the system may be unpredictable.

Limit scope

The scope in which a service can be advertised and discovered can also be pre-defined by attributes to the service information. The service need only be advertised within a certain scope of the system, which can reduce the advertisement and data maintenance workload. The search for a service can also be limited to a certain scope avoiding unnecessary discovery processes. However, pre-knowledge about the service and its requests are needed to achieve optimisation. Mismatches between the scope limitation of a service and of a request may result in the low success rate of the service discovery.

4. Performance Evaluation

In the previous sections a homogeneous agent hierarchy which has a service discovery capability has been described. This is used to give a model of the resource management infrastructure, which is a large-scale distributed software system with a highly dynamic behaviour. However, performance evaluation of such a system is a difficult task, because the system behaviour will become complex when different optimisation strategies are used. In this section, a modelling and simulation environment is introduced to aid the performance evaluation process. A simplified model of the resource management infrastructure is given as a case study and simulation results are included to show the impact of the choice of the performance optimisation strategy on the system performance.

4.1. Modelling

There are four kinds of information that effect the system performance and must be defined in the performance model. These include: the agent hierarchy, the services, the requests, and the optimisation strategies. The modelling and simulation environment provides graphical interfaces for the user to perform the modelling activity at both the agent level and the system level. The only components that exist in the model are agents, so agent-level modelling can be used to define all the model attributes for the simulation. However, system-level modelling is also necessary for modelling agent mobility, service and request distribution, and so on. A system-level strategy definition can affect all of the agents in the model and ease the modelling process.

Table 1. Example model

Agents	Upper Agent
gem	-
sprite~0……sprite~49	*gem*
tup~0……tup~49	*sprite~9*
cola~0……cola~49	*sprite~19*
tango~0……tango~49	*sprite~29*
pepsi~0……pepsi~49	*sprite~39*

(a) Agents

Name	Relative Performance	Freq	Lifetime	Scope	Dist (%)
HPC	1000	5	Unlimited	Top	20
HPC	600	10	Unlimited	Top	40
HPC	200	20	Unlimited	Top	60

(b) Services

Name	Relative Performance	Freq.	Scope	Dist. (%)
HPC	100	5	Top	80
HPC	300	10	Top	60
HPC	500	20	Top	40
HPC	800	40	Top	20
HPC	1000	60	Top	10

(c) Requests

The attributes of an example model are shown in Table 1. This is composed of about 250 agents, each representing a high performance computing resource that may provide a computing capability with a different performance. These agents are organised in a hierarchy, which has three layers. The identity of the root agent is *gem*. There are 50 agents registered to *gem*, four of which each also have 50 lower agents. The hierarchy is illustrated in Table 1(a).

To simplify the modelling processes, we define the services and requests in the agents at the system level. The name of the services and requests are all *HPC*, but with different relative performance values. In practical systems, service performance can be any value such as execution time, memory size, and so on, which can be predicted using the tools such as PACE [15]. The frequency value of the service, 5, for example, means the service performance will change between 0 and the performance value once every 5 steps during the simulation. The frequency value of the request, 5, for example, means a request will be sent once every 5 steps during the simulation. A step can be designed as an arbitrary number of seconds. In practical systems, the frequency values must be monitored while the system is operational. The performance optimisation strategies of the lifetime and scope limitations are not used in the model. The distribution value is used to define how many agents will be configured with the corresponding service or request. The simulator will choose agents randomly to be configured with these system level definitions before simulation begins.

Table 2. Performance optimisation strategy

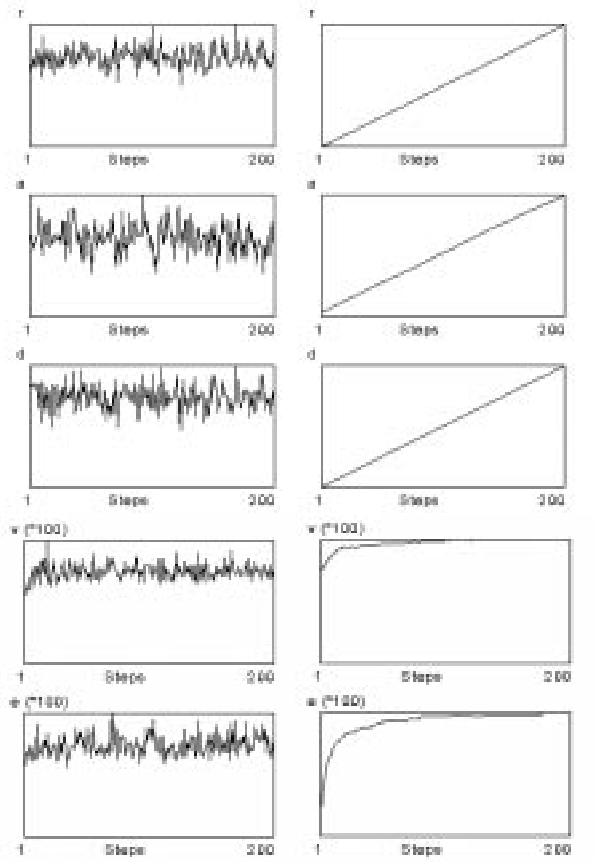
	Experiment Number					
Optimisation Strategy	1	2	3	4	5	6
Using C_ACT		✓	✓	✓	✓	✓
Using L_ACT			✓	✓	✓	✓
Using G_ACT				✓	✓	✓
Updating L_ACT*					✓	✓
Updating G_ACT*				✓	✓	✓
Advertising L_ACT			✓	✓	✓	✓
Multicasting L_ACT						✓

* Here the updating frequency was once every 10 steps.

Finally, the model must define how each agent uses the cache, local, and global knowledge to optimise the performance. In this case study six experiments have been considered, each of which has the same configurations as described in Table 1, but has different optimisation strategies as described in Table 2. To simplify the experiments, we only define the strategies at the system level, which means all of the agents in the model must use the same performance optimisation strategies. A mixture of optimisation strategies is possible but is not considered in these experiments. In the simulation results included in Section 4.2, a comparison of the different strategies is given by considering their impact on the system performance.

4.2. Simulation

When the simulation begins, a thread is created to calculate the statistical data step by step. The phase for request sending and the service discovery is the key part of the whole simulation process. The simulation environment can show the results in multiple views, including a step-by-step, accumulative view etc. For example, Figure 2 shows the simulation results for experiment 4. The number of requests (*r*), the connections for advertisements (*a*), the connections for discovery (*d*), the discovery speed (*v*), and the system efficiency (*e*) are all shown. We assume that the load balancing and discovery success rate are not critical in this study. Attention is given to the discovery speed and the system efficiency.



(a) Step-by-step view (b) Accumulative view

Figure 2. Simulation views

The Simulation results for all of the experiments are summarised in Table 3. Each of the six situations are described in detail below.

Table 3. Simulation results

Metric	Experiment Number					
	1	2	3	4	5	6
r	12296	12355	12576	12560	12645	11715
a	0	0	5604	8051	10172	285148
d	65595	51113	7435	6901	6910	7056
v	0.18	0.24	1.69	1.82	1.82	1.84
e	0.18	0.24	0.96	0.84	0.74	0.04

Note: All values are accumulative results after 200 steps.

In the first experiment, no strategies are selected. Each time the request arrives, a lot of connections must be made and traversed in order to find the satisfied service. In this situation, the discovery speed and system efficiency are both rather low.

In the second, the cache is used in each agent, which needs no extra data maintenance and improves the discovery speed and system efficiency a little. This is because the dynamics of the services reduce the effects of the cached information and so becomes unreliable.

L_ACT is added in each agent in the third experiment. Each time the service performance changes, the corresponding agent will advertise the change upward in the hierarchy. This adds additional data maintenance workload to the system, which decreases the discovery workload extremely. So the discovery speed and the system efficiency are all improved.

G_ACT is also added in the fourth experiment. Each agent will get a copy of the L_ACT of its upper agent once every 10 simulation steps, which will add additional data maintenance workload. From the simulation results, we can see this improves the discovery speed further. But the system efficiency decreases a little because of the additional data maintenance.

Another maintenance of the L_ACT is added in the fifth experiment. Each agent copies the L_ACTs of its lower agents once every 10 steps. This doesn't improve the discovery speed any more and only adds more data maintenance workload, which decreases the system efficiency further.

Another maintenance of the G_ACT is added in the sixth experiment. The change of the L_ACTs will be passed to the G_ACTs of the lower agents. This improves the discovery speed only a little, but adds further data maintenance workload, which decreases the system efficiency extremely.

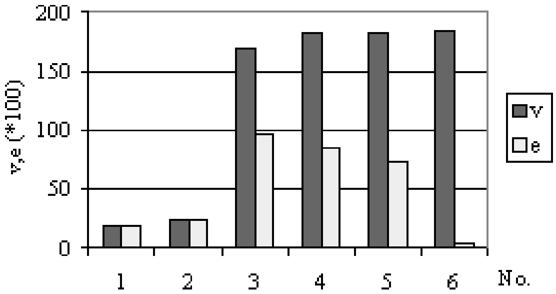


Figure 3. Choice of optimisation strategies

The impact of the choice of the optimisation strategies on the discovery speed and the system efficiency, is shown clearly in Figure 3. It can be seen that the fourth experiment has a good balance between the discovery speed and the system efficiency for this example model. It has a higher discovery speed in comparison to the third, with only slight lower system efficiency.

Changing the G_ACT update frequency will also change the performance of the model. Figure 4 shows the relation between the G_ACT update frequency and the system performance. In these experiments, the strategies that are used are all the same as described in the fourth experiment of Table 2. The only difference is the G_ACTs in the agents are updated with different frequencies, which may lead to differences in the amount of system workload for service advertisement. The best trade-off between discovery speed and system efficiency is once every 20 simulation steps in this example model.

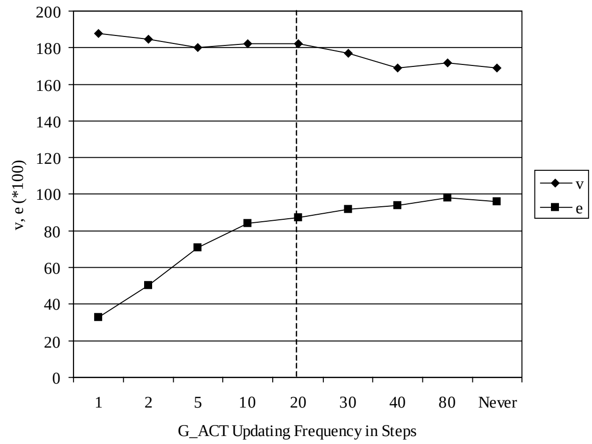


Figure 4. Choice of G_ACT update frequency

In summary, the example model should use all of the C_ACT, L_ACT, and G_ACT. L_ACT should be maintained by the real-time service advertisement. The G_ACT should be maintained by updating once every 20 steps. In fact, the performance of the example model can be improved further if using agent level modelling.

Different agents can use a mixture of different strategies to achieve higher performance of the whole system. This is not discussed in detail here.

5. Conclusions

The resource management system in the grid computing environment will be a large-scale distributed software system with high dynamics. In this work, we have developed a homogeneous agent-based hierarchical model to meet the requirements of the scalability. We also abstract the resource management, scheduling and allocation into the processes of the service advertisement and discovery. The performance issues arise from the high dynamics of the system. There are several common performance optimisation strategies can that be used in this system. However, the evaluation of their impacts on the system performance is difficult. In this work a modelling and simulation environment has been implemented to aid the performance evaluation process. A case study is given and the simulation results show how to examine the different metrics for the system to order to achieve higher performance.

Our future work will focus on the implementation of a resource management infrastructure using the methodology of the agent hierarchy and the service discovery. The infrastructure will be integrated with the PACE functionality [15], which can be used to predict the service performance information dynamically. At the meta level, the performance optimisation strategies of the agents should have the ability to be adjusted according to real-time performance evaluation results produced by the modelling and simulation environment introduced above. The overall aim is to keep the whole infrastructure more efficient to fulfil the functions of resource management.

References

[1] K. Amold, B. O'Sullivan, R. Scheifer, J. Waldo and A. Woolrath, "The Jini™ Specification", Addison Wesley, 1999.

[2] M. Baker, R. Buyya, and D. Laforenza, "The Grid: International Efforts in Global Computing", to appear in ACM Computing Surveys, 2000.

[3] J. Bray, C. Sturman, "Bluetooth: Connect Without Cables", Prentice Hall, 2000.

[4] J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, "Modeling of ASCI High Performance Applications Using PACE", in Proceedings of 15th Annual UK Performance Engineering Workshop, Bristol, UK, pp. 413-424, 1999.

[5] J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, "Performance Modeling of Parallel and Distributed Computing Using PACE", in Proceedings of 19th IEEE International Performance, Computing and Communication Conference, Phoenix, Arizona, USA, pp. 485-492, 2000.

[6] J. Cao, D. J. Kerbyson, and G. R. Nudd, "Dynamic Application Integration Using Agent-Based Operational Administration", in Proceedings of 5th International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology, Manchester, UK, pp. 393-396, 2000.

[7] H. Casanova, and J. Dongarra, "Applying NetSolve's Network-Enabled Server", IEEE Computational Science & Engineering, Vol. 5, No. 3, pp. 57-67, 1998.

[8] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems", in Proceedings of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[9] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, "Resource Management in Legion", Future Generation Computer Systems, Vol. 15, No. 5, pp. 583-594, 1999.

[10] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations", in Proceedings of 6th IEEE Symposium on High-Performance Distributed Computing, pp. 365-375, 1997.

[11] I. Foster, and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Morgan-Kaufmann, 1998.

[12] N. R. Jennings and M. J. Wooldridge (eds), "Agent Technology: Foundations, Applications, and Markets", Springer-Verlag, 1998.

[13] W. Leinberger, and V. Kumar, "Information Power Grid: The New Frontier in Parallel Computing", IEEE Concurrency, Vol. 7, No. 4, pp. 75-84, 1999.

[14] H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, and S. Sekiguchi, "Utilizing the Metaserver Architecture in the Ninf Global Computing System", in Proceedings of High-Performance Computing and Networking , Lecture Notes on Computer Science 1401, Springer-Verlag, pp. 607-616, 1998.

[15] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox, "PACE – A Toolset for the Performance Prediction of Parallel and Distributed Systems", International Journal of High Performance Computing Applications, Special Issues on Performance Modelling – Part I, Sage Science Press, Vol. 14, No. 3, pp. 228-251, Fall 2000.

[16] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", in Proceedings of 7th IEEE International Symposium on High Performance Distributed Computing, Chicago, Illinois, July 1998.

[17] S. H. Russ, K. Reece, J. Robinson, B. Meyers, R. Rajan, L. Rajagopalan, and C. Tan, "Hector: An Agent-Based Architecture for Dynamic Resource Management", IEEE Concurrency, April-June, pp. 47-55, 1999.

[18] W. Yeong, T. Howes, and S. Kille, "Lightweight Directory Access Protocol", RFC 1777, Draft Standard, 1995.